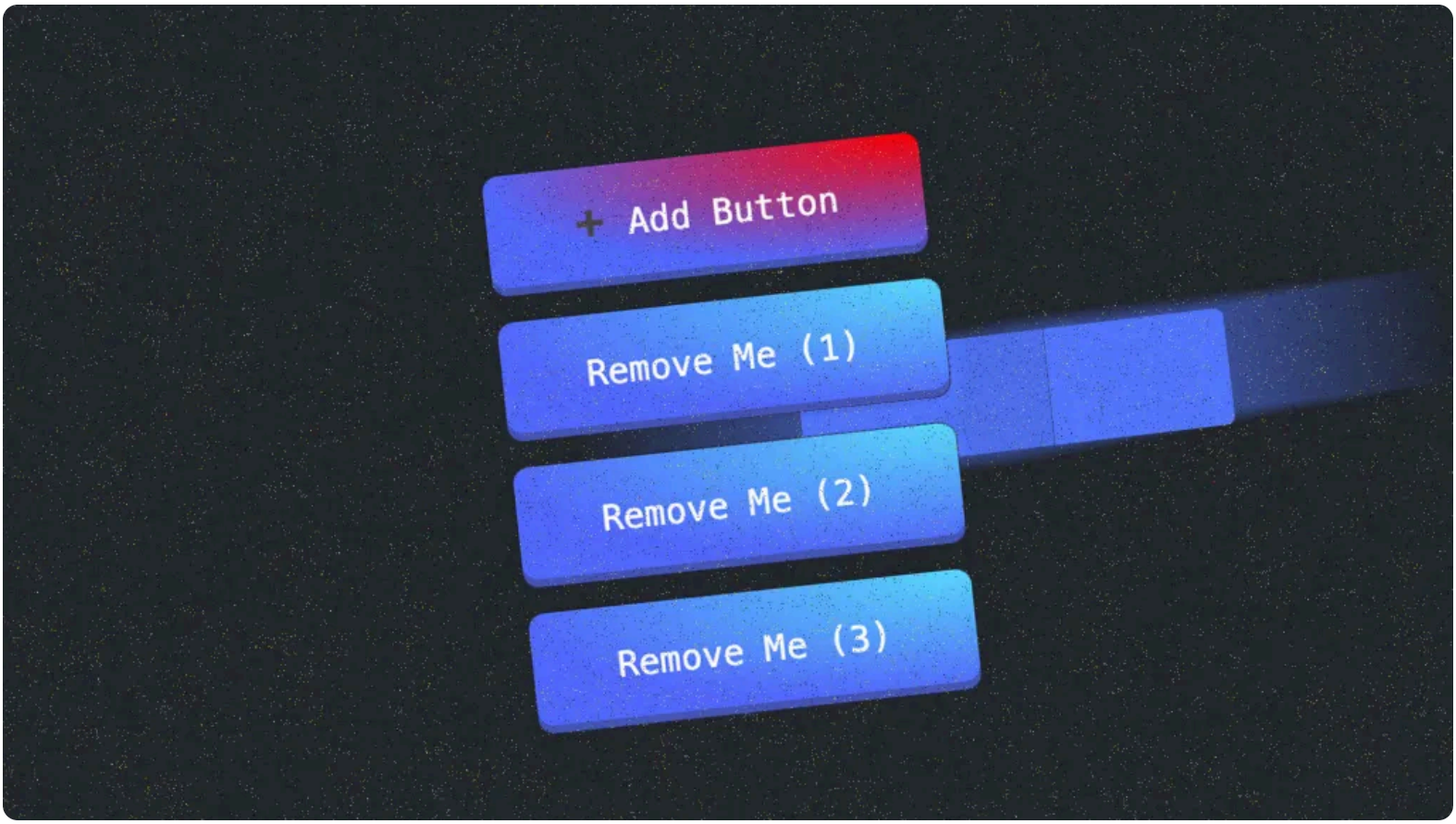
/ BLOG



React’s ViewTransition Element

CHRIS COYIER ON JANUARY 30, 2026

As a bit of a connoisseur of View Transitions and user of React, I’m naturally interested in the fact that React now has a `<ViewTransition>` element it ships directly (in a “Canary” pre-release).

TABLE OF CONTENTS

Did you know?

I wanna take a look at it, but to start, let's... *not* use it. View Transitions are a feature of the web platform itself, not specific to any framework. So React can't really stop us from using them. And it's not entirely weird just do it.

Using View Transitions in React (Classic Style?)

The same-page View Transitions API (the one most relevant for React, as opposed to multi-page View Transitions), is largely this:

JAVASCRIPT

```
document.startViewTransition(() => {
  // change DOM here
});
```

But changing the DOM is... React's job. It doesn't really love it when you do it yourself. So instead of doing any DOM manipulation directly ourselves, we'll do something React-y instead like update state.

JAVASCRIPT

```
import React, { useState } from "react";

export default function DemoOne() {
  const [buttonExpanded, setButtonExpanded] =
useState(false);

  const toggleButton = () => {
    document.startViewTransition(() => {
      setButtonExpanded(!buttonExpanded);
    });
  };

  return (
    <button
```

```
      className={`button ${buttonExpanded ?
"expanded" : ""}`}
      onClick={toggleButton}
    >
      Button
    </button>
  );
}
```

The visual part will be handled by CSS. The state change changes as a class, and the classes change the look.

CSS

```
.button {
   /* button styles */

   &.expanded {
     scale: 1.4;
     rotate: -6deg;
   }
}
```

Getting Ready to use `<ViewTransition>`

The element is only in the “Canary” build of React at the time of this writing, meaning you’d have to install that specifically like:

```
npm install react@canary
```

So your `package.json` would list `canary` as the version.

JSON

```
{
  "dependencies": {
    "react": "canary",
    "react-dom": "canary"
  }
}
```

Or if you’re using React client-side you could have your imports mapped to a CDN URL. Like this if you’re using an import map.

HTML

```
<script type="importmap">
{
  "imports": {
    "react": "https://esm.sh/react@canary",
    "react-dom": "https://esm.sh/react-dom@canary"
  }
}
</script>
```

Using `<ViewTransition>` in React

Now we can import `ViewTransition` itself and use it as a JSX element, along with it's buddy `startTransition`.

JAVASCRIPT

```
import React, { startTransition, ViewTransition }
from "react";

function App() {
  const [buttonExpanded, setButtonExpanded] =
useState(false);

  const toggleButton = () => {
    startTransition(() => {
      // do something that changes the DOM but,
like, in a React-y way.
      setButtonExpanded(!buttonExpanded);
    });
  };

  return (
    <main>
      <ViewTransition>
        <button
          className={`button ${buttonExpanded ?
"expanded" : ""}`}
          onClick={toggleButton}
        >
          Button
        </button>
      </ViewTransition>
    </main>
  );
}
```

The same CSS as above would apply, as all we're doing is toggling a class on a button. But note we're not using like `.classList.toggle("expanded")` as that's a direct DOM method, we're letting React go

through a re-render cycle (or however you say it) and handling that itself.

So… They Both Work Fine?

They sure do. In these limited demos, anyway. So what gives? They even can work together on the same page just fine.

One pretty minor thing is that you'll need to apply a CSS `view-transition-name` yourself on anything you're using `document.startViewTransition` with, while `<ViewTransition>` applies a `view-transition-name` automatically for you. That's a little tiny bonus for `<ViewTransition>`.

The "I Hate This" Part of Me

Part of me doesn't like this at all. React isn't really giving us all that much. It's not making this stuff any easier, it's just making us do it in a way that doesn't disrupt how the framework works. If we spend a lot of time learning this (and there is plenty to learn!) it's not particularly transferrable knowledge to anywhere else.

The "OK, Fine" Part of Me

React wants to handle the DOM for you, and it always has since Day One. That's what you're buying, and because of that, you have to buy into letting it do

certain things for you. This means using `<ViewTransition>`, presumably, is going to do things like “automatically coordinate the transition with its rendering lifecycle, Suspense boundaries, and concurrent features” and do things like batch updates, prevent conflicts, mange nesting, and whatever complicated crap you and I don’t want to think about.

Also, things are *a little bit* more “declarative” in that you’re being very specific about where you are applying the wrapping `<ViewTransition>` element, which may jive with people’s mental model better. But you still need to call `startTransition` so it’s still fairly imperative too, and I can imagine in more complex nested UIs, it’ll be a bit confusing to figure out where best to orchestrate all this.

I admit I kinda like the very specific attributes like `enter` and `exit` on the `<ViewTransition>` element, which maps to “bring you own” CSS view transition classes. This is more straightforward to me than the `:only-child` technique of figuring it out for yourself.

So, I’ll leave you with a demo like that:

Start Learning Today →

Leave a Reply

Comment *

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

$966,000

Frontend Masters donates to open source projects through thanks.dev and Open Collective, as well as donates to non-profits like The Last Mile, Annie Canons, and Vets Who Code.

Contact: support@frontendmasters.com
